Kyle Marlia-Conner

6/15/2025

Module 3

Version Control Guidelines in DevOps

Version control is like keeping a detailed inventory of your belongings when you have a younger sibling who loves playing with your things. Imagine you’ve carefully arranged your Lego sets, and your little brother comes in, moves pieces around, or even takes some away and hides them. Without a system to track what was where or how things were before, you’d have no idea how to fix it or return it to its original state. Version control does the same for code; it keeps snapshots of every change, so if something breaks or goes missing, you can return to a working version. It also shows you *who* made changes and *why*, which is especially helpful when working in teams. Just like how you’d want to know what your siblings have changed, developers need to see what their teammates have modified to avoid chaos. In modern software development, Version Control Systems (VCS) are essential for tracking code changes, enabling collaboration, and managing workflows. With the rise of DevOps, the importance of well-defined version control guidelines has only grown.

To better understand what version control can look like in today’s DevOps environment, this will be comparing three sources that are each being about a year apart from each other: [a 2023 Modern Requirements blog](https://www.modernrequirements.com/blogs/version-control-best-practices/), [Medium’s 2024 “Git Strategies & Best Practices,”](https://medium.com/@aa.adnane/git-strategies-best-practices-for-effective-version-control-2a420d7a6776) and [Atlassian’s Git tutorials](https://www.atlassian.com/git/tutorials/what-is-version-control) ( as well as a supplementary article on [Trunk-based development](https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development)). This paper aims to highlight current guidelines supporting teams in this ever-evolving industry by utilizing their recommendations in modern DevOps and identifying common themes.

Comparing the Three

All three sources share the foundational importance of consistent branching models, meaningful commit messages, and collaborative workflows, though each approach this differently. While the Medium article concentrates on advanced Git workflows optimized for team efficiency, highlighting models like Git Flow, Feature Branching, and Trunk-Based Development. It underscores the need to tailor version control strategies to the size and structure of the development team. Atlassian’s Git tutorial, on the other hand, offers an overview accessible to both beginners and experienced developers. It explains various branching strategies and merge types while encouraging habits such as early commits, peer reviews, and transparency. Meanwhile, Modern Requirements connects version control to traceability and requirements management, strongly emphasizing documentation, regulatory compliance, and aligning code changes with business objectives.

Contrasts (Git Flow vs. Trunk-Based)

While the core principles of version control remain consistent across these sources, there are differences in how certain practices are prioritized. One key example is Git Flow, a once-popular branching strategy that is now often viewed as overly complex for modern CI/CD environments. Medium and Atlassian highlight its limitations, particularly in fast-paced DevOps settings where [Trunk-Based Development](https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development) (merging smaller updates to the core branch instead of keeping longer-lived ones) is preferred for its simplicity and faster integration. The Modern Requirements article stands apart in its strong emphasis on documentation and traceability, which are essential in highly regulated industries such as healthcare or finance (and uses the tale of Knight Capital as an example). Another area of consensus among all three sources is the discouragement of long-lived feature branches. These are increasingly seen as detrimental due to the higher risk of merge conflicts and delayed integration, with the modern trend favoring short-lived branches that are merged frequently into the mainline to maintain code cohesion and reduce disruption.

A Refined List of Essential Version Control Guidelines

Using these three sources and keeping current DevOps best practices in mind, I believe the following guidelines are needed for maintaining a streamlined, collaborative, and scalable development workflow:

1. Use a Consistent Branching Strategy  
   Whether it is Trunk-Based Development or Git Flow, consistency is key. Choose a model that aligns with you and your team.
2. Commit Early and Often  
   Frequent commits make tracking changes, identifying bugs, and understanding project history easier. Early commits also enable earlier integration and feedback.
3. Write Clear, Descriptive Commit Messages  
   Each commit message should explain *what* was changed and *why* it was changed. This helps improve traceability, peer reviews, and speeds up debugging.
4. Keep Branches Short-Lived  
   Avoid long-running branches by merging work frequently. This reduces merge conflicts and smooths integration, especially in CI/CD environments.
5. Require Peer Code Reviews Before Merging  
   Code reviews not only improve code quality but also encourage knowledge sharing. All three sources recommend this practice as critical for DevOps success.
6. Use Tags for Release Versions  
   Tagging release points allows for rollback, testing, and reproducibility. It is essential for environments where deployment stability is a priority.
7. Integrate Version Control with CI/CD  
   Automate builds, tests, and deployments by linking your version control system with your CI/CD tools.
8. Trace Changes to Requirements or Tasks  
   Whenever possible, link commits to tickets, requirements, or user stories. This improves traceability, accountability, and transparency.
9. Secure Access and Enforce Permissions  
   Use repository permissions and branch protections to prevent unauthorized changes. This helps maintain code integrity and project stability.

Conclusion

Version control guidelines are the foundation for effective collaboration, traceability, and automation in DevOps. While practices like Git Flow and long-lived branches may be losing relevance in fast-paced environments, other practices, such as short-lived branches, descriptive commits, and CI/CD integration, are becoming standard. By adopting a consistent and thoughtful version control strategy, teams can accelerate development, reduce errors, and foster a DevOps culture of transparency and continuous improvement.

References:

*What is version control: Atlassian Git Tutorial*. Atlassian. (n.d.-b). https://www.atlassian.com/git/tutorials/what-is-version-control

*Gitflow workflow: Atlassian Git Tutorial*. Atlassian. (n.d.-b). https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

*Trunk-based development*. Atlassian. (n.d.-c). https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development

Adnane Arharbi, E. (2024, May 19). *Git strategies: Best practices for effective version control*. Medium. https://medium.com/@aa.adnane/git-strategies-best-practices-for-effective-version-control-2a420d7a6776

Satpathy, A. (2023, August 21). *Version control best practices*. Modern Requirements. https://www.modernrequirements.com/blogs/version-control-best-practices/